VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Miloš Kubáček

Bakalářská práce

Vedoucí práce: Ing. Lenka Skanderová, Ph.D.

Ostrava, 2021

Abstrakt

Tento dokument popisuje odbornou praxi ve firmě VRK plus s.r.o. V úvodní části je pospána samotná firma a použité technologie. Dále je popsána přesná náplň vykonávané práce. Jedním z cílů této práce bylo nastudovat a vyvinout systém pro vytváření úloh umožňující běh na pozadí pro portál Liferay s použitím již vytvořeného frameworku. Druhým cílem bylo prostudování a nahrazení současného e-mailového systému používaného v jednom z firemních portálů. K vytvoření nového e-mailového systému má být použit již zmíněný systém na vytváření úloh na pozadí.

Klíčová slova

Java EE, Liferay, JavaServer Faces, Background task, Mandrill

Abstract

This document describes a professional practise in VRK plus s.r.o company. In the introduction part is described the company itself and used technologies. Afterwards is described the exact content of the work performed. One of the goals of the work was to study and create a system for creating tasks running on background for Liferay portal with usage of an already existing framework. The second goal was to study and replace the current email system used in one of the company's portals. The already mentioned system for background tasks should be used in the creation of the new email system.

Keywords

Java EE, Liferay, JavaServer Faces, Background task, Mandrill

Poděkování

Rád bych poděkoval v první řadě svému konzultantovi Ing. Martinu Vaňkovi a vedoucí své práce Ing. Lence Skanderové, Ph.D. Dále bych rád poděkoval ostatním pracovníkům firmy a rodině.

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
EVSRŠ	–	Elektronizácia vzdelávacieho systému regionálneho školstva
HTML	–	Hypertext Markup Language
HTTP	–	Hypertext Transfer Protocol
ISIS	–	Internetový školní informační systém
JSF	–	JavaServer Faces
JSON	–	JavaScript Object Notation
MVC	–	Model-View-Controller
SMTP	–	Simple Mail Transfer Protocol
ŠVP	–	Školní vzdělávací programy
UML	–	Unified Modeling Language
URL	–	Uniform Resource Locator
WML	–	Wireless Markup Language
XHTML	–	Extensible Hypertext Markup Language
XML	–	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V této bakalářské práci je popsána náplň mé praxe v rámci firmy VRK plus s r.o. Ve zmíněné firmě pracuji od roku 2016, tehdy jsem začínal pouze jako brigádník a pracoval jsem na navádění dat do jednotlivých firemních systémů. Postupně jsem se dostal k pozici testera firemních aplikací, abych zaplnil pomyslnou mezeru v procesu vývoje softwarového produktu. Zabýval jsem se přípravou testovacích scénářů a s přibývajícími znalostmi jsem vyvíjel automatické testy pro portál pohodlne.info, kterým jsem se zabýval v rámci své odborné praxe.

Náplní mé praxe bylo nastudovat a vytvořit systém pro vytváření kontrolovatelných operací běžících na pozadí v rámci portálu. Další částí odborné praxe bylo zavést vytvořený systém do provozu a nahradit stávající řešení pro odesílání e-mailů skrze portál prostřednictvím služby Mandrill.

Kapitola 2

VRK plus s.r.o

Společnost VRK plus s r.o. se zabývá tvorbou vlastních informačních systémů a desktopových aplikací určených pro školy, sportovní kluby či volnočasové organizace. Firma sídlí v Ostravě a v době této bakalářské práce má dvanáct zaměstnanců. Firma nabízí své služby pro školy a organizace po celé České republice i na Slovensku.

2.1 Historie firmy

Společnost VRK plus s r.o. byla založena v roce 1993. Na počátku se zabývala vytvářením informačních systému a webových stránek.

V roce 2003 vznikl produkt *Internetový školní informační systém* (zkr. ISIS), který se stal základem pro další produkty. Systém ISIS je zaměřen na vysoké či vyšší odborné školy, které ho mohou využívat k evidenci uchazečů a studentů či definici studijních plánů dané školy. Na tento produkt dále navazovala aplikace SMILE, která umožňuje školám po celé České republice vytvářet školní vzdělávací programy (zkr. ŠVP) odpovídající předpisům *Ministerstva školství, mládeže a tělovýchovy*, dále slouží k jejich rozložení do jednotlivých čtvrtletí školního roku [1].

Z aplikace SMILE poté vychází projekt *Elektronizácia vzdelávacieho systému regionálneho školstva* (zkr. EVSRŠ), který byl vyvíjen společně se slovenskou firmou DWC Slovakia a.s. pod záštitou slovenského ministerstva školství.

2.2 Pohodlne.info

Projekt pohodlne.info vznikl v roce 2016 sloučením několika aktivit firmy VRK plus s r.o. Dříve samostatné portály detske-tabory.info a ceske-skoly.info byly zastřešeny novým portálem, který rozšířil možnosti pro stávající uživatele. Do portálu pohodlne.info byla navíc začleněna aplikace SMILE KLUB, která slouží k evidenci členů. Tímto aplikace získala zcela nový vzhled a byla integrována do celku pohodlne.info.

Portál pohodlne.info umožňuje uživatelům pohodlně vyhledávat informace o dostupných sportovních oddílech, zájmových kroužcích, zajímavých jednorázových akcích či jiných volnočasových aktivitách v jejich regionu. Dále jim portál umožňuje samotnou registraci na události a jejich placení. Cílovou skupinou uživatelů jsou organizace pořádající volnočasové aktivity buď jednorázové (např. tábor, workshop, školení) či dlouhodobé (např. členství v klubu). Systém pomáhá také rodičům a dětem tyto aktivity nalézt, přihlásit se na ně a mít přehled o evidovaných údajích.

Volnočasovými institucemi mohou být různé spolky nebo organizace. v současné době využívá portál více než 2400 registrovaných organizací, golfových klubů, tanečních škol, kroužků bojových umění a dalších zájmových kroužků.

Pro pořadatele volnočasových aktivit nabízí portál možnost spravování událostí, evidenci účastníků a evidenci dlouhodobých členů. Součástí správy událostí je i možnost řízení plateb, kde účastníci mohou vyřizovat a platit registrace s využitím platební brány. Zveřejnění události na portálu je zdarma, ale portál nabízí i placené tarify, které s sebou přinášejí některé výhody v podobě větší kapacity účastníků a členů, dále technickou podporu a možnostmi rozšířené ekonomiky.

Uživatelé mají možnost se na portálu zaregistrovat, což jim umožní přednastavit filtrování nabízených organizací či akcí. Pokud organizace, kterou uživatel navštěvuje, využívá evidenci členů, může si registrovaný uživatel zobrazit svou kartu člena. V rámci členské karty si může pro jednotlivé události a organizace prohlédnout, jaké informace o něm pořadatel eviduje [2].

Kapitola 3

Použité technologie

V této kapitole budou popsány významné technologie, se kterými jsem se během praxe setkával. Jednotlivé technologie jsou zmíněny v pořadí podle své důležitosti.

3.1 Liferay

Liferay Portal (dále portál) umožňuje vytvořit systém, který je nezávislý na aplikačním serveru. Jedná se o službu založenou na jazyce Java, která se může používat jako hlavní podnikový portál. Umožňuje vytvořit rozsáhlé uživatelské rozhraní, které může být napsáno v mnoha jazycích fungujících pod jednou střechou. Tato vlastnost je zajištěna použitím *pluginů*[1], které mohou portál vylepšovat a rozšiřovat. Portál se skládá z jednotlivých menších funkčních komponent, které se nazývají portlety (viz sekce 3.2). Kompletní systém zaštiťuje jednotlivé portlety, umožňuje jejich zobrazení, správu a seskupení. Portál podporuje všechny mainstreamové aplikační servery, operační systémy a databáze.

Dostupné verze portálu

Liferay Portal byl distribuován ve dvou verzích. Verze Community Edition (CE) byla bezplatná a dostupná volně ke stažení. Podpora této verze je zajištěna pomocí rozsáhlých fór a jiných volně dostupných dokumentací. Druhou verzí byla komerční Enterprise Edition (EE). Oproti bezplatné verzi byl kladen větší důraz na bezpečnost, výkon a stabilitu portálu. Verze EE nabízela technickou podporu přímo od společnosti Liferay nebo některého z ověřených partnerů. Od verze 7 se ale tyto dvě edice přestaly používat. Byly nahrazeny verzemi Portal, která je bezplatná, a placenou verzí Digital Experience Platform (zkr. DXP) [3]. Portál pohodlne.info je postaven na verzi 6.2. CE.

[1] Z anglického plugin (v překladu zásuvný modul). Jedná se o softwarové rozšíření jiné aplikace, kterou je rozšířena/nahrazena funkčnost původní aplikace.

3.2 Java Portlet

Portlety jsou funkční komponenty zajišťující komunikaci mezi webovou aplikací a portálem. U informačního systému běžícího na portálu tvoří portlety prostředníka mezi uživatelským rozhraním prezentační vrstvy a vrstvou doménové logiky informačního systému. Portlety svou funkčností připomínají technologii servletů a to tím, že reagují na uživatelské požadavky. Liší se ale tím, že na rozdíl od servletů, které generují celou stránku, portlety generují pouze fragment dokumentu nezávisle na URL. Již zmíněné fragmenty poté obsahují kus kódu ve značkovacím jazyce (jako HTML, XHTML, WML).

Stránka portálu se skládá z několika samostatných portletů, kdy každý generuje část dokumentu dané stránky. Důležitou součástí technologie portletů je portletový kontejner, který odpovídá za životní cyklus jednotlivých portletů. Řeší jejich incializaci, zpracovává vyvolané požadavky na jednotlivé komponenty, jejich vykreslování a v neposlední řadě i ukončení instancí, u kterého dochází k uvolnění zdrojů serveru.

3.3 JavaServer Faces

JavaServer Faces (zkr. JSF) je technologií Java Enterprise Edition. Hlavní využití má JSF právě ve vytváření webových portálů. JSF umožňuje psaní uživatelského rozhraní pomocí XML tagů a pomocí standardních Java Bean. Standard Java Bean se vyznačuje tím, že všechny vlastnosti třídy jsou privátní a jsou spravovány metodami `get()` a `set()`. Dále obsahuje bez-parametrický konstruktor, což umožňuje jednoduché vytvoření instance s následným přiřazením všech hodnot a implementuje rozhraní `Serializable`.

Technologie JSF je postavená na návrhového vzoru Model-View-Controller (zkr. MVC) [4]. Jednotlivé komponenty technologie JSF, jsou propojené pomocí *Expression Language* s klasickými Java třídami, které jim poskytují data, přístup ke službám nebo do jiných systémů [5].

3.4 JSF - Beans

Managed beans používané technologií JSF jsou speciálním případem klasických Java Bean. Tyto managed beany[2] jsou mocným nástrojem pro vývoj pomocí JSF technologie. Umožňují jednoduché propojení komponent uživatelského rozhraní a tříd informačního systému. K propojení komponent se používá unikátní název managed beany, které je psáno pomocí již zmíněného *Expression Language*, které umožňuje volání funkcí a proměnných dané beany. Vytvořená beana obsluhuje danou komponentu tím, že zpracovává vyvolané události nebo vyplňuje její vlastnosti.

[2] V praxi došlo k počeštění anglického beans na beany.

Použití vlastnosti Scope

K vytvoření instancí třídy skryté za beanou se váže tzv. scope (v překladu rozsah), který určuje, kdy je instance vytvořena a kdy zaniká. Velikost rozsahu se liší podle vlastní definice. Existuje šest definic rozsahu, zde jsou vypsány ty nejpoužívanější:

- **RequestScoped -** podle doby zpracování HTTP požadavku.
- **ViewScoped -** podle zobrazení stránky obsahující beanu.
- **ApplicationScoped -** po celou dobu spuštění webové aplikace.
- **CustomScoped -** vlastní definice.

3.5 Primefaces

Primefaces je open-source knihovna, která rozšiřuje prvky uživatelského rozhraní původních JSF komponent. Dále nabízí nové pokročilejší komponenty, nástroje a pomocné funkce pro práci s JSF, například komponenty jako ukazatel průběhu (angl. progress bar), *drag&drop* nebo samostatný textový editor a další. Všechny prvky této knihovny jsou popsány v rozsáhlé volně dostupné dokumentaci. To poté umožňuje vytvořit webovou stránku s jednotnou stylizací, s využitím moderních webových technologií jako HTML5 s možností responzivního designu.

3.6 Mandrill

Mandrill je placená e-mailová služba, která nabízí rozsáhlé API pro práci s e-mailovou komunikací. Poskytuje analýzu e-mailového provozu a umožňuje přidávat tagy, či metadata k jednotlivým zprávám. Tyto tagy mohou být využity ke sledování zpráv (to zda jsou otevřeny apod.) nebo například k filtraci zpráv ve statistikách. Metadata umožní přidání vlastních parametrů k obsahu zprávy, jako například identifikační číslo uživatele. Metadata jsou ukládány ve formátu JSON.

Použití událostí webhook

Další možností je použití tzv. *webhooků*, které podle své definice mohou vyvolávat události na straně portálu. Možnými událostmi může být úspěšné odeslání, zpoždění, otevření zprávy, označení zprávy jako nevyžádaná pošta, apod. V neposlední řadě umožňuje vytváření vlastních šablon pro hromadné či opakované posílání e-mailů.

3.7 Git

Git je open-source verzovací systém, který umožňuje přehlednou správu verzí softwarového produktu. Systém Git byl původně přizpůsoben operačnímu systému Linux, ale později byl implementován pro ostatní Unixové operační systémy a poté i pro Windows. Systém Git umožňuje paralelně vyvíjet jednotlivé části softwaru, rozdělit ho do několika větví a poté je znova spojit dohromady. Tím ulehčuje práci v týmu, takže je možné aby každý programátor mohl vyvíjet paralelně a nedochází ke kolizím.

Kapitola 4

Náplň odborné praxe

Firma měla zájem o prozkoumání možností vytváření úloh na pozadí a nahrazení a inovaci stávajícího e-mailového systému používaného v rámci portálu pohodlne.info. S portálem Liferay ani s technologií JSF jsem v rámci studia bohužel nepracoval, proto bylo nutné si dané technologie nastudovat, abych mohl začít pracovat na hlavní náplni své praxe.

Praxe byla rozdělena do dvou samostatných fází, které na sebe navazovaly. V první fázi jsem se zabýval vlastním systémem pro vytváření úloh na pozadí portálu. Druhou fází se stalo nahrazení původního e-mailového systému. Časová náročnost jednotlivých částí je znázorněna v Tabulce 4.1.

Samotný portál poskytuje více řešení. Server Tomcat, na kterém je Liferay portál spuštěn, nabízí možnost využití tzv. `ThreadPoolu`, který umožňuje spuštění vláken nezávisle na webovém portálu. Ovšem toto řešení neumožňuje žádnou možnost monitorování a podrobnější správy jednotlivých vláken. Dalším řešením může být použití rozhraní `PortalExecutorFactory`, které je součástí portálu. Toto rozhraní nabízí vytvoření samostatného exekutoru s použitím rozhraní `Runnable`, využívaného u klasických Java vláken.

Společně s konzultantem jsme se ale rozhodli použít Background Task framework, který narozdíl od přechozích řešení nabízí rozsáhlé rozhraní a nástroje pro práci s úlohami na pozadí, vytvořený přímo společností Liferay. Toto API umožňuje správu vytvořených úloh, společně se zpracováním výjimek. Dále nabízí předáváním souborů a dat k úloze a v neposlední řadě integraci s Liferayovskou

Tabulka 4.1: Vyjádření časové náročnosti

Vykonaná práce	**Počet dnů**
Instalace a seznamování se s vývojovým prostředím	2
Studium potřebných technologií a jejich aplikace	8
Fáze první: Vytvoření prototypu pro úlohy na pozadí	20
Fáze druhá: Nahrazení e-mailového systému	20
Celkem	50

technologií `ServiceBuilder`. Výhodou této integrace je lepší správa jednotlivých tříd, které jsou součástí portálu. Umožňuje vytvořit servisní vrstvu mezi uživatelským rozhraním a databází.

4.1 Fáze první: Vytvoření prototypu pro úlohy na pozadí

Mým úkolem bylo navrhnout a vytvořit systém s využitím již zmíněného frameworku, který by bylo možné využít pro různé oblasti portálu pohodlne.info. Proto jsem se rozhodl nejdříve vytvořit funkční prototyp systému pro práci s úlohami. Prototyp by měl implementovat tyto funkce:

- Zobrazení právě probíhajících úloh.
- Správu jednotlivých úloh.
- Zobrazení historie úloh s jejich výsledky.

Na tento prototyp jsem navázal při druhé části své praxe a vytvořil nový systém pro e-mailovou komunikaci v rámci portálu pohodlne.info. Využití prototypu spočívá v tom, že při odeslání formuláře bude vytvořena úloha na pozadí se všemi potřebnými daty k odeslání e-mailu.

Pod pojmem úloha je možné představit si cokoliv, co je třeba vykonat. Úloha symbolizuje veškerá data, se kterými během vykonávání pracujeme, i samotné vykonávání. Příkladem úlohy může být například výpočet složité matematické rovnice, generování platebních údajů z obsahu košíku, vytvoření zálohy databáze nebo v našem případě odeslání e-mailu. Úlohy mohou obsahovat vstupní data či parametry, ale není to nutností.

4.1.1 Background Task Framework

V této kapitole bude popsán základní návrh systému pro práci s úlohami na pozadí jako součást portálu Liferay, který jsem v rámci prototypu používal.

Hlavní entitou je třída `BackgroundTask`, která drží veškeré informace o vykonávané úloze. Obsahuje informace o tom, kdo, kde a kdy vytvořil danou úlohu a v jakém se právě nachází stavu. Dále si drží tzv. `taskContextMap` (dále jen kontext-mapa), která uchovává všechny nutné proměnné pro správné vykonání úlohy. Kontext-mapa je hlavní způsob, jak dostat potřebná vstupní data do samotného vykonávajícího vlákna. Další speciální součástí `BackgroundTask` jsou její přílohy (angl. attachments), u kterých portál nabízí přidání souboru pomocí `PortletFileRepository` frameworku, který v rámci portálu ukládá jednotlivé soubory pro každou úlohu samostatně.

Pro samotné vykonání úlohy na pozadí je připravené rozhraní `BackgroundTaskExecutor` (dále jen exekutor) obsahující základní metodu `execute()`, která vykonává úlohu. Tu je nutné při implementaci rozhraní nahradit právě kódem, který chceme, aby se ve vláknu provedl. Metoda `execute()` by se dala přirovnat ke klasické metodě `run()`, která je součástí Java vláken. Metoda `execute()` vrací instanci `BackgroundTaskResult` určující, zda úloha doběhla úspěšně, či nikoli. Pro zachycení

vyvolané výjimky v hlavní větvi exekutoru je vytvořena metoda `handleException()`. Tato metoda automaticky nastavuje úlohu za nezdařenou a ukončí ji. K návrhu třídy exekutoru se ještě vztahuje atribut `serial` s datovým typem `boolean`, který určuje, zda mohou být úlohy stejného typu vykonávány současně, nebo za sebou.

Důležitou součástí frameworku je systém monitorování průběhu vykonávání úloh. Ten využívá systém `Liferay Message Bus`. Tento systém umožňuje posílat a číst zprávy napříč portálem. Ke čtení vytvořených zpráv o úlohách se používá rozhraní `BackgroundTaskStatusMessageTranslator` (dále jen překladač), které čte zprávy ze sběrnice a převádí je do instance `BackgroundTaskStatus` Tím umožňuje získat a číst informace o úloze. Podle potřeb systému je možné přizpůsobit překladač specificky pro daný exekutor.

Samozřejmě pro správné fungování exekutoru není nutné žádné zprávy o stavu vykonávané úlohy posílat, tím pádem není nutné implementovat překladač. Jak bylo zmíněno výše, instance úlohy si drží informace o svém současném stavu či výsledku. Zprávy slouží pouze k detailnější kontrole a informování jiných pracujících vláken, které mohou monitorovat proces běžících úloh [6].

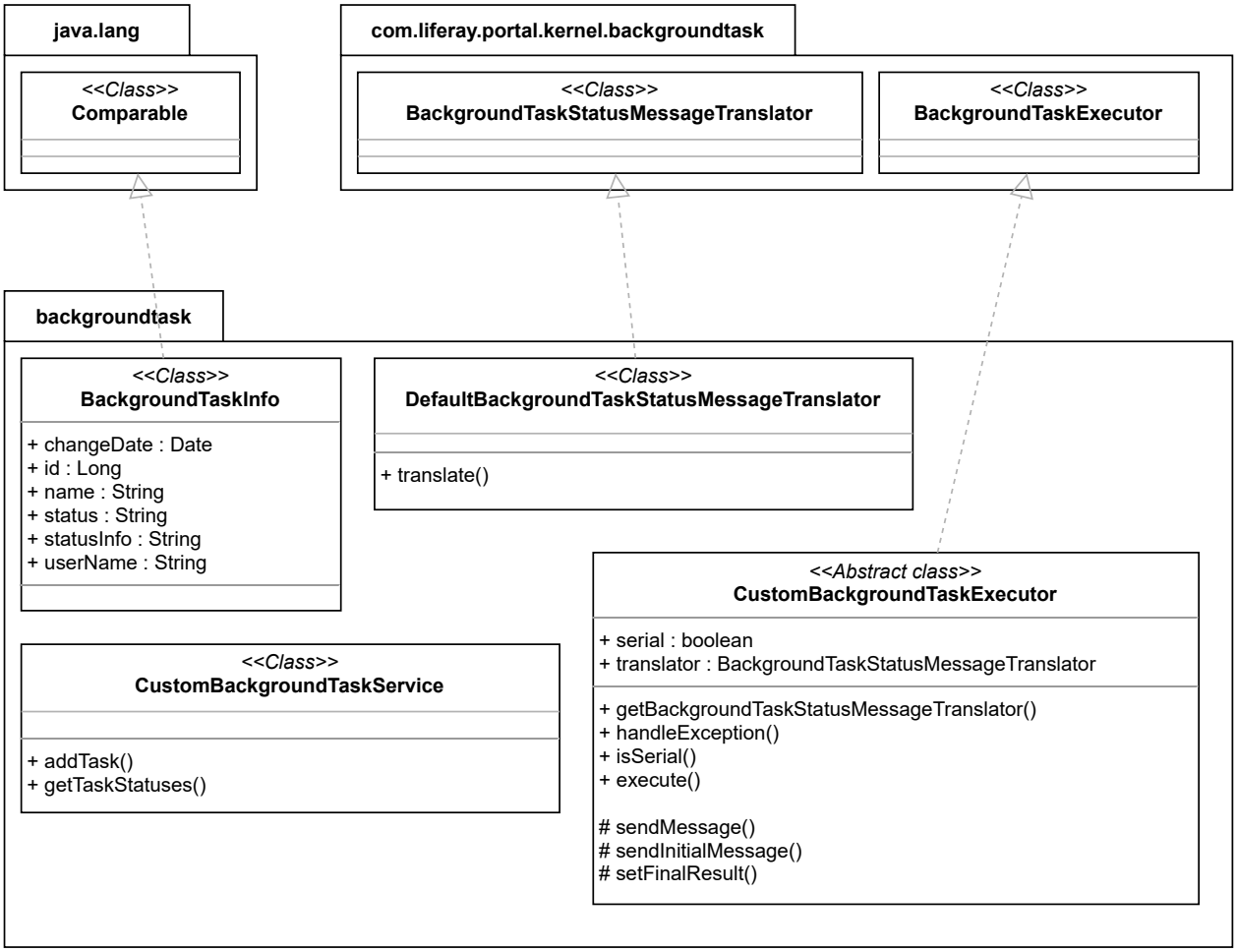
4.1.2 Návrh prototypu

Jak už bylo zmíněno v úvodu této kapitoly, zadáním mé praxe bylo vytvořit systém pro práci s úlohami na pozadí v rámci portálu. Systém by měl ulehčit manipulaci s nabízeným frameworkem a přizpůsobit ho k tomu, aby bylo možné ho využít napříč celým portálem. Dále by měl umožnit vytvořené úlohy monitorovat a získávat informace o již ukončených úlohách.

Zjednodušený návrh vytvořeného systému se všemi použitými třídami je znázorněn na Obrázku 4.1. Následující kapitoly se postupně zabývají všemi důležitými komponentami nového návrhu a následně nalezenými problémy při jeho testování.

4.1.2.1 Vytváření úloh

Mým hlavním záměrem při vytváření návrhu bylo vytvořit takový systém, který by umožnil jakémukoliv programátorovi pracujícím na portálu pohodlne.info vytvořit úlohu na straně serveru bez podrobných znalostí nabízeného frameworku. Proto jsem se rozhodl vytvořit dvě vlastní třídy, přesněji servisní třídu `CustomBackgroundTaskService` (dále jen servisní třída) pro vytváření úloh a abstraktní třídu `CustomBackgroundTaskExecutor`, která by navazovala na původní exekutor z nabízeného frameworku. Společně by tyto dvě třídy tvořily jednoduchý minimální základ, který by umožnil práci s úlohami. Programátor by se nemusel dále o nic víc starat. Jediné, čím by se musel zabývat, je vytvoření kontext-mapy a vytvoření vlastního exekutoru podle abstraktu z `CustomBackgroundTaskExecutor` (k přesnému využití této třídy se dostanu později v sekci 4.1.2.2). Poté by mohl přes instanci servisní třídy zavolat funkci `addTask()`, která přijímá právě kontext-mapu, název třídy executoru a případné přílohy k dané úloze. Servisní třída podle instance



Obrázek 4.1: Zjednodušený návrh prototypu

portálu doplní potřebné parametry a za pomocí frameworku vytvoří novou úlohu se zadanými vstupními daty a nechá ji spustit přes zadaný executor.

Zjednodušení pomocí servisní třídy spočívá v tom, že při vytváření úlohy pouze za pomocí frameworku je nutné k vytvoření úlohy dodat i informace o právě přihlášeném uživateli, jeho vybrané organizaci a instanci `ServiceContext`, která drží informace o právě prováděném požadavku z portletu. Díky servisní třídě může programátor pouze poslat data důležitá k provedení úlohy a nemusí se zaobírat jejím samotným vytvořením.

K ulehčení správného vytvoření jednotlivých úloh bylo vytvořeno pomocné rozhraní `ExecutorDefinition` (dále jen definice), které je možné implementovat v rámci systému. Význam tohoto rozhraní je vytvoření kontext-mapy na míru danému executoru. To umožňuje definovat, jaké parametry kontext-mapa musí obsahovat, případně zde může dojít k jejich validaci. Poté je možné instanci definice použít k vytvoření nové úlohy. Toto rozhraní nabízí další prostor pro případnou validaci vstupních dat daných úloh.

4.1.2.2 Monitorování vytvořených úloh

Zbývající část mého návrhu přináší systém monitorování jednotlivých úloh. Mým cílem bylo vytvořit jednotný formát, který by se dal aplikovat na každý vytvořený typ úlohy. Po dohodě s vedoucím vývoje jsem vytvořil datovou třídu `BackgroundTaskInfo`, která zastupuje instanci jedné z vytvořených úloh. Tato třída se využívá pro přenos dat mezi jednotlivými objekty. Obsahuje identifikační číslo úlohy, její status a datum poslední změny. Dále obsahuje jméno uživatele, který úlohu vytvořil. Třída `BackgroundTaskInfo` je vytvořena hlavně pro vývojářský komfort, ale nabízí dle potřeby prostor pro další rozšíření. S touto třídou je poté lehká manipulace, protože neobsahuje žádné atributy navíc. Pokud by bylo třeba zobrazit další informace o úloze, jako například přesné identifikační číslo uživatele, či jeho organizace, je nutné pomocí frameworku získat celý záznam o úloze přes její identifikační číslo. Třída `BackgroundtaskInfo` ve struktuře listu je využita jako návratová hodnota metody servisní třídy, která má za úkol získat informace o všech vytvořených úlohách. Datovou strukturu listu jsem si vybral kvůli snadnému filtrování a procházení.

Druhé využití datové třídy `BackgroundTaskInfo` bylo posílání stavových zpráv z hlavního vlákna executoru. Vlastní executor jsem rozšířil tím, že jsem navrhl metodu `sendMessage()` pro odesílání zpráv přes `Message Bus`. Tato metoda vytvoří instanci `BackgroundTaskInfo` s informacemi o úloze a aktuálním stavem úlohy během vykonávání. Tuto instanci poté převede do zprávy, kterou pošle do sběrnice. Servisní třída podle použitého executoru vytáhne ze sběrnice informace o všech vytvořených úlohách, které vykonával daný exekutor.

Zjednodušení pro programátora spočívá v tom, že se nemusí zabývat vytvořením zprávy se všemi informacemi a poté následným odesláním přes framework. Díky rozšíření exekutoru stačí pouze poslat hodnotu stavu a jeho případnou bližší specifikaci.

Za zmínku dále stojí vytvoření vlastního překladače, který jsem implementoval z původního rozhraní frameworku (`DefaultBackgroundTaskStatusMessageTranslator`, viz Obrázek 4.1). Tento překladač je nastavený tak, aby převáděl informace ze zpráv na instance třídy `BackgroundTaskStatus` podle navržených parametrů datové třídy.

Dalším rozšířením exekutoru bylo vytvoření ukončovací metody `setFinalResult()`, která vrací instanci `BackgroundTaskResult`, kterou musí každá exekuce skončit. Navíc tato metoda pošle poslední zprávu o dokončení úlohy.

Jak už jsem zmínil dříve, není nutné, aby programátor nějaké zprávy odesílal, ty jsou využity pouze pro přesnější monitorování vykonávaných úloh, ale na samotném vykonávání úloh se nijak nepodílejí. Kdyby o hledané úloze neexistovaly žádné zprávy, je metoda servisní třídy navržena tak, aby získala z instance úlohy pouze defaultní stav (viz Ukázka kódu 4.1).

```
public final List<BackgroundTaskInfo> getTaskStatuses(Class<? extends
    CustomBackgroundTaskExecutor> executorClassname) throws SystemException {
    ArrayList<BackgroundTaskInfo> backgroundTaskInfos = new ArrayList<>();

    List<BackgroundTask> tasks = BackgroundTaskLocalServiceUtil.getBackgroundTasks
        (getGroupId(), executorClassname.getName());

    for (BackgroundTask task : tasks) {
        BackgroundTaskStatus status = BackgroundTaskStatusRegistryUtil.
            getBackgroundTaskStatus(task.getBackgroundTaskId());

        if (status == null) {
            backgroundTaskInfos.add(new BackgroundTaskInfo(task, task.
                getStatusLabel(), task.getStatusMessage()));
        } else {
            backgroundTaskInfos.add(new BackgroundTaskInfo(task, status.
                getAttribute("status").toString(), status.getAttribute("statusInfo"
                ).toString()));
        }
    }
    return backgroundTaskInfos;
}
```

Listing 4.1: Získání informací o úlohách

4.1.3 Nalezené problémy

Během implementace a testování mého návrhu jsem se potýkal s několika komplikacemi. Jednotlivé problémy jsem se rozhodl zmínit a popsat v následujícím textu.

4.1.3.1 Serializace kontext-mapy

Jedna z komplikací, která stojí za zmínku, je problém s kontext-mapou. Původní framework je navržený tak, že při vytváření úlohy dojde ke serializaci kontext-mapy a k uložení dat do databáze. Proto musí všechny parametry kontext-mapy implementovat rozhraní `Serializable`. Což pro jednoduché datové formáty není problém. Většina základních datových typů rozhraní `Serializable` implementuje, ale například obecná datová struktura `java.util.List` nikoli, proto je nutné použít například třídu `ArrayList`, která je serializovatelná.

Proto se při předávání parametrů nutných pro vykonání úlohy musí myslet na to, aby se předávaly pouze objekty splňující tuto podmínku. Tato podmínka ovšem může limitovat přenos instancí složitějších tříd, které po serializaci nelze znova sestavit do správného objektu pro použití v metodě exekutoru. Proto je nutné zajistit získání instancí jednotlivých objektů jinak, například načtením z databáze přes jejich identifikační číslo.

4.1.3.2 Problém vrstvy exekutoru

Jedním z požadavků na návrh exekutoru byl, že bude použit k vykonání doménové logiky (podle třívrstvé architektury). To sebou přináší druhý problém nalezený při testování prototypu. v rámci vykonávání úlohy není možné použít některé funkce specifické pro prezentační vrstvu. v našem případě to znamená, že nejde použít metody JSF frameworku pro získání aktuálního kontextu spojeného s HTTP požadavkem, se kterým většina stávajícího kódu počítá. Proto je nutné případný kontext pro vykonávání úlohy, jako například informace o přihlášeném uživateli, poslat do exekutoru právě skrze kontext-mapu.

4.1.3.3 Přidávání příloh

Další z problémů, se kterým jsem se setkal, bylo přidávání příloh k úlohám. Framework nabízí metody pro jejich přidání, ale to je možné pouze u již vytvořených úloh. Problém je v tom, jak je samotný framework navržen, tudíž když je úloha vytvořena, tak se automaticky spustí její vykonávání. Komplikace nastává ve chvíli, kdy se vykonávání úlohy v rámci vlákna exekutoru provede dříve, než jsou samotné přílohy zpracovány portálem a jsou přiřazeny k úloze. Framework počítá s tím, že vytváření úloh je prováděno v servisní vrstvě v rámci jedné transakce. Proto jsem při implementaci metody na přidání nových úloh využil rozhraní `TransactionInvoker` (viz Ukázka kódu 4.2), které je součástí portálu. Řešení umožnilo vytvořit úlohu a přiřadit ji nahrané soubory

v rámci jedné transakce a až poté došlo ke spuštění vlákna exekutoru vykonávající úlohu. To zajistí, že při vykonávání exekuce jsou dané přílohy jistě k dispozici.

```
public final void addTask(Map<String, Serializable> taskContextMap, Class<?
    extends CustomBackgroundTaskExecutor> taskExecutorClass, List<File> files) {
    try {

    ...

    TransactionInvokerUtil.invoke(
        defaultTransactionAttributeRollbackOnAnyException(), () -> {
        BackgroundTask task = BackgroundTaskLocalServiceUtil.addBackgroundTask(
            userId, groupId, StringPool.BLANK, new String[]{servletContextName},
            taskExecutorClass, taskContextMap, serviceContext);

        if (files != null) {
            for (File file : files) {
                if (file != null) {
                    BackgroundTaskLocalServiceUtil.addBackgroundTaskAttachment(
                        userId, task.getBackgroundTaskId(), file.getName(), file);
                }
            }
        }
        return null;
    });
    } catch (Throwable e) {
            System.out.print(e.getCause().toString());
    }
}
```

Listing 4.2: Použití transakce k přiřazení příloh úloze

4.2 Fáze druhá: Nahrazení e-mailového systému

Druhou fází mé odborné praxe bylo nahrazení stávajícího systému pro odesílání e-mailů, který je součástí portálu pohodlne.info. Nejprve bylo mým úkolem si stávající systém projít, zjistit jeho funkce, vstupní data a principy, se kterými pracuje. Následně jsem mohl začít s návrhem svého systému, který by využíval vytvořený prototyp a původní systém by mohl plnohodnotně nahradit. Jedním z bodů zadání bylo, aby nový systém nepřišel o žádné původní funkce a nabídl prostor

pro implementaci nových. Nový systém má také odstranit současné nedostatky původního systému, kterými se budu zabývat v sekci 4.2.2.

Původní e-mailový systém nabízí uživatelům tyto funkce:

- Hromadné odeslání e-mailů vybraným členům, popřípadě účastníkům své akce.
- Filtrování kontaktů členů podle definovaných kontaktních skupin.
- Generování obsahu podle jednotlivých příjemců vybráním jedné ze šablon, které jsou pro danou organizaci vytvořeny.
- Nahrávání příloh, které jsou s e-mailem spojeny.
- Přidání e-mailové adresy odesílatele podle přihlášeného uživatele.
- Uložení vytvořené e-mailové komunikace do databáze.

4.2.1 Původní systém pro e-mailovou komunikaci

V této kapitole bude popsán základní návrh původního systému, ze kterého jsem následně vycházel. Systém využívá službu Mandrill pro samotné odesílání e-mailů. E-mailová komunikace, kterou je možné prostřednictvím služby odeslat, musí obsahovat tyto parametry:

- Informace o odesílateli jako jeho e-mailovou adresu a případně jméno.
- Předmět zprávy a její obsah.
- Seznam příjemců.

U e-mailové adresy odesílatele je třeba zkontrolovat, zda má adresa jednu z povolených domén, které jsou definovány v nastavení této služby, jinak k odeslání e-mailu nedojde. Příjemci se ve službě Mandrill i v původním systému dělí na hlavní příjemce a poté na vedlejší příjemce, kterým je možné poslat kopii, či skrytou kopii zprávy. Nepovinnou součástí e-mailové komunikace jsou její přílohy nebo případné obrázky, které mohou být součástí těla zprávy.

Systém by se dal rozdělit do těchto tří základních komponent: hlavní e-mailový formulář, třída `EmailViewBean` a třída `EmailCommunicationBean`. Hlavní e-mailový formulář nabízí již zmíněné funkce jako zvolení si e-mailu odesílatele, vybrání si kontaktů nebo vyplnění obsahu, popřípadě vybrání šablony. Dále umožňuje uživateli vyplnit vedlejší příjemce a nahrát přílohy.

Na tento formulář přímo navazuje třída `EmailViewBean`, která prostřednictvím technologie JSF zajišťuje jeho obsluhu. Tato třída drží všechny hodnoty z formuláře a provádí jejich validaci. k inicializaci instance view třídy je třeba definovat seznam členů, přesněji příjemců e-mailu. Následně je z tohoto seznamu vytvořený seznam všech kontaktů členů. z těch je poté vzat seznam kontaktních

typů, ze kterých může uživatel vybírat[1]. Pokud některý z vybraných členů nemá žádný kontakt ze zvolených kontaktní typů, je uživateli nabídnuto daného člena odstranit z hlavního seznamu. Během inicializace je vytvořen základ e-mailové komunikace v podobě entity `MemberCommunication`, která reprezentuje e-mailovou komunikaci pro jednoho člena. Instance `MemberCommunication` reprezentují záznam komunikace se členem, které jsou pomocí servisní třídy ukládány do databáze. Součástí inicializace je vytvoření seznamu šablon, který je generován dle právě zvolené organizace uživatele.

Při odeslání formuláře zahájí třída `EmailViewBean` odesílání. Pro každého z členů je vytvořena instance e-mailové komunikace, která je naplněna základními údaji a kontakty člena podle zvolených kontaktních typů. K uložení kontaktu se využívá entita `MemberCommunicationEmail`, která drží informace o příjemci, jeho jménu a druhu příjemce. Tyto instance jsou vloženy do seznamu a přiřazeny do instance e-mailové komunikace. Dále jsou stejným způsobem přiřazeni i vedlejší příjemci. Pokud je vybrána jedna z nabízených šablon, je vygenerován specifický obsah, který je přiřazen do komunikace. Vytvořená e-mailová komunikace je zaslána do instance třídy `EmailCommunicationBean`, která zodpovídá za převedení e-mailové komunikace do formátu využívaného službou Mandrill.

4.2.2 Problémy původního systému

V této kapitole budou popsány hlavní problémy původního systému, které má nový návrh řešit. Jednotlivé problémy jsou pomyslně seřazeny podle závažnosti a požadavků na systém. Zmíněné problémy spolu mohou souviset.

4.2.2.1 Zastaralý kód

Samotný kód systému pro e-mailovou komunikaci by se dal označit za takzvaný *legacy code* (česky starý nebo zastaralý kód). Původní kód už neodpovídá aktuálním požadavkům na systém. Důvodů pro nahrazení původního systému může být hned několik. Při vytváření systému se nebralo v potaz provádění odesílání e-mailů na pozadí. Třída `EmailViewBean` byla velmi konkrétní a umožňovala odeslat e-mail jen a pouze přes původní formulář. Dále třída `EmailViewBean` naplňovala funkce více vrstev, takže bylo těžké rozhodnout, která část komunikuje s uživatelským rozhraním a která část zodpovídá za samotné odesílání (podrobněji v sekci o problému dvou vrstev).

Dalším z problémů systému bylo, že neměl přesně definované vstupy a výstupy. Všechny vstupy a proměnné byly přímo získávány z obsluhující třídy `EmailViewBean` a nebylo jasné, která data jsou doopravdy potřebná pro vykonání odesílání. Další výtkou k původního kódu je absence podrobnější dokumentace jednotlivých metod. Kvůli tomu není na první pohled jasné, jak původní kód funguje.

[1]Kontaktními typy může být například kontaktní adresa člena, adresa jeho rodiče nebo dítěte.

4.2.2.2 Nedostupnost portálu

Jeden z hlavních problémů původního systému byl ten, že po odeslání formuláře probíhalo odesílání e-mailů v hlavním vykonávacím vlákně. Toto znemožnilo přihlášenému uživateli další použití portálu, dokud odeslání nebylo ukončeno. Počet příjemců není omezen, tudíž samotné odesílání mohlo zabrat i několik minut, přičemž byla webová stránka portálu pro uživatele nedostupná.

4.2.2.3 Problém dvou vrstev

Původní systém sice pracoval správně, ale jeho návrh neodpovídá ověřeným principům při vývoji informačních systémů. Například třída `EmailViewBean` nedodržuje princip třívrstvé architektury. Tím je přesněji myšleno, že současně naplňovala funkce prezentační i aplikační vrstvy. Obsluhovala hlavní formulář, zpracovávala jím vyvolané události a zobrazovala případné výjimky uživateli, což je náplní prezentační vrstvy. Ovšem samotná třída poté i zpracovávala vstupy a chystala e-mailovou komunikaci k jejímu odeslání, při které docházelo ke kontrole uživatelských kontaktů podle firemní logiky, což už je náplní aplikační vrstvy.

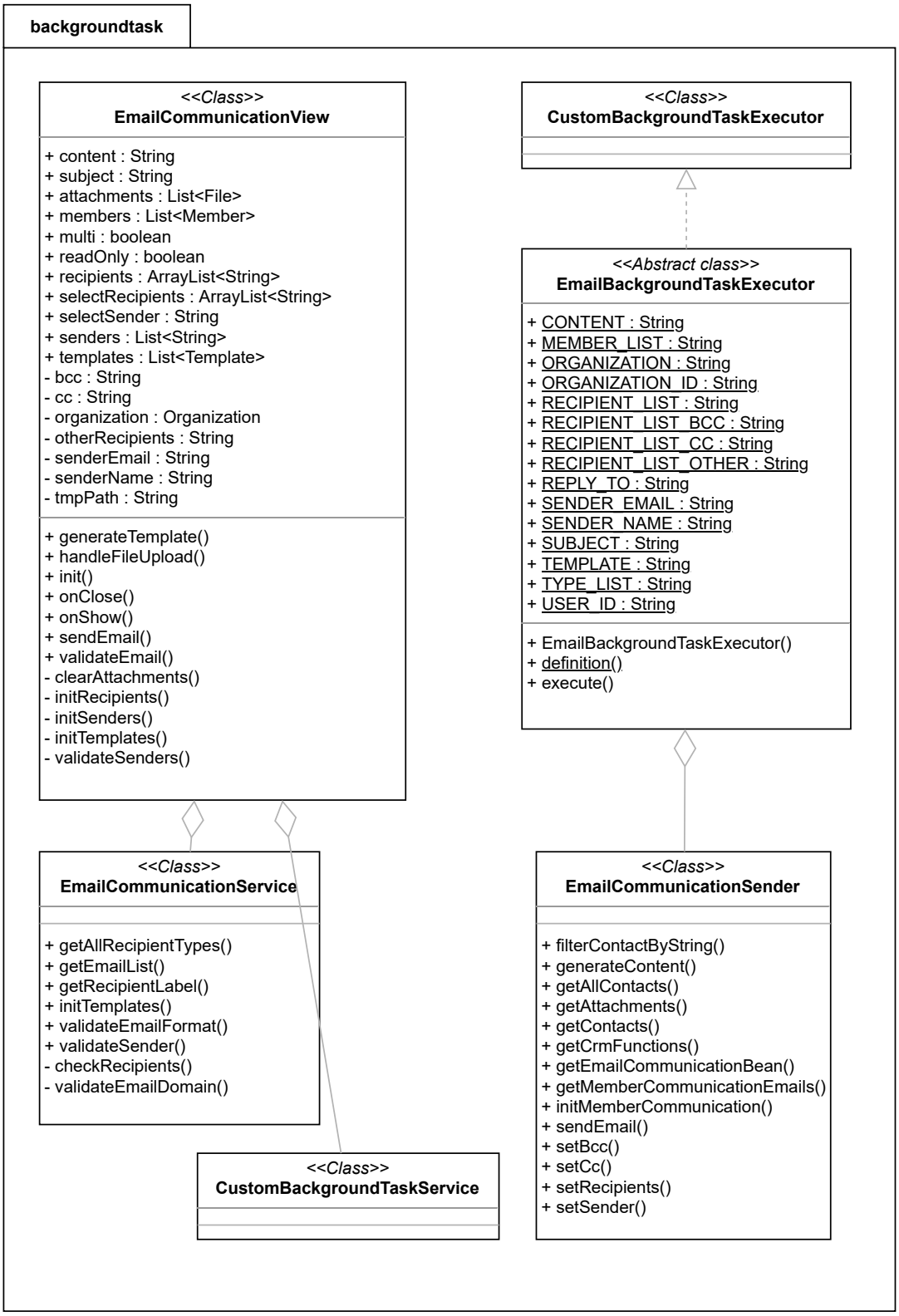
4.2.2.4 Duplicitní kód

Dalším problémem této třídy je její nepřehlednost. Část kódu, která zodpovídá za vytvoření členských kontaktů, je naprosto stejná jako získání kontaktů pro členovy další kontakty (jeho rodiče nebo děti). Ovšem tyto dvě části jsou dány za sebe, bez použití speciální metody. Tudíž v hlavní funkci pro odesílání existuje takřka duplicitní kód, na kterém na první pohled nejde poznat, co dělá. Navíc stejný kód je znova zkopírován a použit, aby mohlo dojít k získání všech možných kontaktních typů z jednotlivých kontaktů. To samé platí o generování obsahu ze šablony. Pro uživatelskou kontrolu je šablona vygenerována a zobrazena ve formuláři, ale během odesílací metody je tento kód znovu zkopírován a použit pro generování opravdového obsahu pro příjemce, znova bez dílčí použití metody.

4.2.3 Nový systém pro e-mailovou komunikaci

Po nastudování původního systému jsem se rozhodl nahradit pouze třídu `EmailViewBean`. Rozhodnutí bylo založené na tom, že třída `EmailCommunicationBean` neobsahovala žádné problémy, tudíž je možné ji v novém systému znovu využít. Vzhled hlavního formuláře jsem se rozhodl taktéž ponechat, protože na něho mohou být uživatelé zvyklí a neobsahuje žádné zbytečné komponenty.

Původní kód byl psán již od rané fáze vývoje portálu a postupným přidáváním funkcí se z něj stal zastaralý nepřehledný kód (viz sekce 4.2.2.1 o zastaralém kódu), u kterého si kolegové nebyli jistí, které části jsou zodpovědné za jaké funkce. Proto jsem kód musel pochopit a rozlišit, které části řeší jakou funcionalitu. Funkčnosti původní třídy jsem rozdělil na dvě hlavní části: obsluha hlavního



Obrázek 4.2: Zjednodušený návrh systému

formuláře a samotné odesílání. To umožňuje vytvořit dvě různé třídy, kdy by každá zajišťovala jednu z funkčních částí. Do systému jsem podle zadání zakomponoval i navržený prototyp.

Při vytváření nového systému jsem postupoval následovně. Vytvořil jsem třídu `EmailCommunicationSender`, která by zastupovala odesílání e-mailu pro jednoho člena. Dále jsem vytvořil view třídu `EmailCommunicationView`, která obsluhuje hlavní formulář podobně jako původní `EmailViewBean` třída. Pro přehlednost nové view třídy jsem vytvořil servisní třídu `EmailCommunicationService`, která získává a validuje potřebná data k formuláři. Zjednodušený návrh s výše zmíněnými třídami nového e-mailového systému je uveden v Obrázku 4.2.

Důležitou součástí náplní mé praxe bylo refaktorování původního kódu a jeho přizpůsobení aktuálním technologiím. K testování správnosti řešení a odesílání e-mailů, jsem používal nástroj FakeSMTP[2], který simuluje vlastní SMTP server na localhost.

4.2.3.1 Třída EmailCommunicationSender

Hlavním úkolem třídy `EmailCommunicationSender` je zajištění e-mailové komunikace podle vstupních dat, které je poté možné odeslat prostřednictvím původní třídy `EmailCommunicationBean`. Třída byla navržena tak, aby zajišťovala správné vytvoření instance `MemberCommunication`. Třída nabízí metody pro její inicializaci a následně všechny ostatní metody přijímají právě instanci e-mailové komunikace, do které ukládají jednotlivé vstupní atributy. Takovýto návrh dovoluje jednotlivé metody volat v libovolném pořadí.

Jednou z nabízených metod je metoda `getContacts()` pro získání hlavních kontaktů daného člena. Další je metoda `getAllContacts()` na získání všech kontaktů vrací hlavní kontakty společně s hlavními kontakty všech členů, kteří mají s daným členem nějaký vztah (viz Ukázka kódu 4.3). K tomu využívá opakované volání první metody. Takovýto návrh metod eliminuje jeden z vytýkaných problémů původního systému o duplicitním kódu (viz sekce 4.2.2.4).

```
public List<ContactHelper> getAllContacts(Member member) throws SystemException {
    List<ContactHelper> memberContacts = this.getContacts(member, ContactHelper.
        RecipientType.MEMBER);

    List<MemberRelation> relations = MemberRelationLocalServiceUtil.getByMember(
        member.getId());
    for (MemberRelation relation : relations) {
        if (relation.isValid(new Date())) {
            if (relation.getMemberChild() == member.getId()) {
```

[2]Dostupný ze stránky http://nilhcem.com/FakeSMTP/.

```
                memberContacts.addAll(this.getContacts(relation.getParent(),
                    ContactHelper.RecipientType.PARENT));
            } else if (relation.getMemberParent() == member.getId()) {
                memberContacts.addAll(this.getContacts(relation.getChild(),
                    ContactHelper.RecipientType.CHILD));
            }
        }
    }
    return memberContacts;
}
```

Listing 4.3: Ukázka získání všech kontaktů člena

Další z metod je filtrace všech kontaktů podle zvolených kontaktních typů. Ta funguje podobně jako v původním systému, při které je vzat první kontakt daného typu a zbývající kontakty stejného typu nejsou použity[3] (viz Ukázka kódu 4.4).

```
public List<ContactHelper> filterContactByString(List<ContactHelper> contacts,
    List<String> types){
    List<ContactHelper> newContacts = new ArrayList<ContactHelper>();
    List<String> usedTypes = new ArrayList<>();

    if(types.size()==0){
        return contacts;
    }
    for(ContactHelper contact : contacts){
        for(String type : types){
            if(contact.getContactType().equals(type) && !usedTypes.contains(type)){
                newContacts.add(contact);
                usedTypes.add(type);
            }
        }
    }
    return newContacts;
}
```

Listing 4.4: Ukázka filtrování kontaktů

[3]Kontaktní adresa člena a kontaktní adresa rodiče, popřípadě dítěte, se počítají jako dva různé kontaktní typy.

Po kompletním vytvoření e-mailové komunikace nabízí třída již zmíněnou metodu `sendEmail()` pro její odeslání prostřednictvím `EmailCommunicationBean`. v této metodě dochází k přiložení případných příloh e-mailu.

4.2.3.2 Vlastní e-mailový exekutor

Po vytvoření třídy `EmailCommunicationSender` bylo dalším krokem vytvoření speciálního executoru pro odesílání e-mailů. Samotný kód nového exekutoru je součástí přílohy A.1. Hlavní vykonávací metodu by se dala rozdělit do tří částí:

- Získání všech potřebných dat z kontext-mapy.
- Transformace získaných dat do potřebných formátů pro e-mailovou komunikaci.
- Samotné odesílání e-mailů.

Jak bylo zmíněno v sekci o vytváření úloh (viz 4.1.2.1), není možné pomocí kontext-mapy poslat neserializovatelné objekty. Proto vstupní data exekutoru musí být zjednodušena. Například seznam členů, kterým má být e-mail odeslán, nemůže být reprezentován přímo jako seznam objektů vlastní třídy `Member`, ale musí být nahrazen seznamem identifikačních čísel jednotlivých členů. Dalším příkladem jsou vedlejší příjemci. Ti jsou zjednodušeni pouze na e-mailové adresy reprezentované jako instance datového typu `String`. Součástí kontext-mapy dále musí být identifikátory uživatele a jeho organizace.

Po získání všech potřebných dat dochází k jejich zpracování. Podle identifikátorů jednotlivých členů je získan seznam instancí třídy `Member`. E-mailové adresy vedlejších příjemců jsou převáděny do instancí `MemberCommunicationEmail`, které jsou využity při odesílání. Z identifikačního čísla uživatele jsou získány informace o právě přihlášeném uživateli, které jsou použity k monitorování průběhu úlohy. V neposlední řadě je identifikační číslo organizace použito ke generování obsahu e-mailu a vytvoření instance třídy `EmailCommunicationBean`.

Při samotném odesílání dochází k postupnému procházení seznamu členů, přičemž je pro každého člena vytvořena instance třídy `MemberCommunication`. Postupně jsou přidány informace o odesílateli, předmětu a obsahu. Pokud je vybrána šablona, tak je vygenerován obsah podle daného člena. Následně jsou získány všechny členovy kontakty, u kterých dojde k filtraci podle zvolených kontaktních typů. Do instance e-mailové komunikace jsou přidání všichni příjemci a dojde k jejímu odeslání. Všechny tyto operace jsou vykonávány s využitím třídy `EmailCommunicationSender`. Na konci každé iterace cyklu dochází k odeslání zprávy o současném stavu úlohy, podle množství odeslaných e-mailů.

4.2.3.3 Formulář, View a Servisní třída

Po vytvoření exekutoru bylo následným krokem převzetí původního hlavního formuláře a k němu vytvoření nové view třídy. Původní formulář je zapsán ve formátu XHTML a jeho komponenty

nebylo třeba upravovat. S využitím technologie JSF-Bean bylo možné připojit jednotlivé komponenty k hodnotám vlastností nové view třídy. Třída `EmailCommunicationView` obsahuje všechny nutné vlastnosti podobně jako původní třída `EmailViewBean`. Tato třída je inicializována při načtení stránky s formulářem a při její inicializaci dochází k nastavení právě zvolené organizace.

Při inicializaci formuláře dojde k zavolání metody `onShow()`, která nastaví hlavní seznam členů a vynuluje vlastnosti jako předmět a obsah e-mailu nebo seznam příloh. Případně při inicializaci dojde k nastavení dalších vstupních vlastností, například předem vybrané šablony. Podle vstupních vlastností jsou vytvořeny seznamy kontaktních typů a nabízených šablon. K získání jednotlivých informací jsou použity metody třídy `EmailCommunicationService`, aby nová view třída zůstala přehlednou.

Součástí této třídy (respektive formuláře) jsou i metody na nahrávání a správu nahraných souborů jakožto příloh e-mailu. Při vybrání nového souboru dojde ke kontrole, zda příloha nepřekračuje povolenou kapacitu. Pokud by ji seznam příloh překračoval, uživatel je upozorněn a k nahrání nového souboru nedojde. U již nahraných příloh má uživatel možnost jejich smazání ze seznamu nebo má možnost jejich stažení k případné kontrole. Při smazání je příloha odebrána ze seznamu view třídy a ze serveru je smazána.

Při odeslání formuláře dojde k zavolání metody `sendEmail()` (viz Ukázka kódu 4.5), která zodpovídá za vytvoření nové e-mailové úlohy. Součástí této metody je validace odesílatele, při které je provedena kontrola formátů a domény, aby nedošlo k vyvolání výjimky až při vykonávání odesílání. Pokud je vybrán členský e-mail a jeho doména neodpovídá jedné ze seznamu povolených, je jako odesílatel nastavena e-mailová adresa zvolené organizace a uživatelův e-mail je nastaven jako e-mail pro odpověď[4]. Dále v rámci této metody dojde ke sestavení kontext-mapy úlohy, do které jsou uloženy všechna potřebná data z formuláře. Zde dochází ke zjednodušení některých vlastností pro přenos do exekutoru. Pokud nedojde při validaci dat k vyvolání výjimky, je tato kontext mapa odeslána třídě `CustomBackgroundTaskService`, ve které dochází k vytvoření samotné úlohy. Současně dojde i k přeposlání příloh, které jsou nahrány prostřednictvím Background Task frameworku k úloze. Při zavření formuláře dojde ke smazání celé složky patřící dané e-mailové komunikaci obsahující dočasně nahrané soubory. K validaci jsou použity metody vytvořené servisní třídy `EmailCommunicationService`.

```
public void sendEmail(){
    // This taskContextMap can be used as transporter to background job
    Map<String, Serializable> taskContextMap = new HashMap<String, Serializable>()
        ;
```

[4]Pokud k této události dojde, je uživatel upozorněn, ale nedojde ke zrušení odesílání.

```
    ...

    if(this.selectTemplate!=null){
        String templateUuId = this.selectTemplate.getUuid();
        taskContextMap.put(EmailBackgroundTaskExecutor.TEMPLATE, templateUuId);
    }
    else {
        taskContextMap.put(EmailBackgroundTaskExecutor.SUBJECT,this.subject);
        taskContextMap.put(EmailBackgroundTaskExecutor.CONTENT, this.content);
    }

    ...

    taskContextMap.put(EmailBackgroundTaskExecutor.ORGANIZATION_ID, this.
        organization.getOrganizationId());
    taskContextMap.put(EmailBackgroundTaskExecutor.USER_ID,
        LiferayPortletHelperUtil.getUser().getUserId());


    try {
        this.backgroundTaskService.addTask(taskContextMap,
            EmailBackgroundTaskExecutor.class, attachments);
    }
    finally{
        this.clearAttachments();
    }
}
```

Listing 4.5: Zjednodušená ukázka sestavení kontext-mapy

4.2.3.4 Monitorování odesílání e-mailu

Pro kontrolu průběhu odesílání jsem vytvořil vlastní webovou stránku s využitím technologie JSF. Na této stránce je právě přihlášenému uživateli zobrazena tabulka obsahující jím vytvořené e-mailové úlohy. Tabulka se průběžně aktualizuje, takže uživatel vidí informace o právě probíhajících úlohách, v jakém jsou stavu a kolik e-mailů už je odesláno. V této tabulce současně vidí i výsledky již dokončených úloh a případné chyby, ke kterým během jejich vykonávání mohlo dojít.

K zobrazení potřebných informací jsou využity instance datové třídy `BackgroundTaskInfo`, které jsou získány metodami servisní třídy `CustomBackgroundTaskService`. Vytvořená stránka je minimalistická, ale ukazuje možnost praktického využití systému, pro kontrolu e-mailového provozu.

4.2.3.5 Analýza zbývajících požadavků

Součástí zadaných požadavků na nový e-mailový systém bylo i rozšíření v podobě lepší kontroly e-mailového provozu. Přesněji je tímto myšleno dávková komunikace s e-mailovou službou, zprostředkování informací o nedoručitelných e-mailech, zařazení do nevyžádané pošty (spam), automatické odhlašování z hromadné pošty a přístup ke statistikám. Tato rozšíření bohužel byla svým rozsahem nad původní plán odborné praxe, proto k jejich implementaci během vykonávání praxe nedošlo.

K rozšíření systému o tyto požadavky by byla použita již zmíněná služba Mandrill. Pomocí Mandrill API by bylo možné monitorovat provoz podle zvolené organizace. Služba dále umožňuje prostřednictvím API či přímo přes webovou stránku služby vytvořit tzv. tag[5] nebo webhook (viz sekce 3.6). Vytvořené tagy mohou sloužit k monitorování jednotlivých zpráv a zjištění jejich aktuálního stavu.

Jak už bylo zmíněno, je možné pomocí webové stránky služby vytvořit vlastní webhook, který při zachycení definované události (například otevření zprávy nebo její zařazení do spamu) pošle HTTP POST požadavek na definovanou stránku. Tělo požadavku je definováno ve formátu JSON a obsahuje informace o typu události a zprávě samotné (viz Ukázka kódu 4.6). Pro vygenerovaný požadavek by byla vytvořená stránka, která by ho dokázala zpracovat například s využitím technologie Servletů. V našem případě by stránka mohla vyvolat událost, která by uživatele upozornila, kdyby některý z e-mailů byl označen jako spam. Tuto událost by bylo možné provázat s monitorováním vykonaných e-mailových úloh podle identifikačního čísla úlohy, které by mohlo být součástí meta-dat zprávy (viz sekce 3.6).

```
{
  "mandrill_events"[
      {
         "event":"spam",
         "msg":{
            "ts":1365109999,
            "subject":"This an example webhook message",
            "email":"example.webhook@mandrillapp.com",
            "sender":"example.sender@mandrillapp.com",
            "tags":[
```

[5]Tagem může být například označení zprávy jako žádost o placení nebo sdělení o nových aktualizacích.

```
            "webhook-example"
         ],
         "opens":[
            {
               "ts":1365111111
            }
         ],
         "clicks":[
            {
               "ts":1365111111,
               "url":"http:\\mandrill.com"
            }
         ],
         "state":"sent",
         "metadata":{
            "user_id":111
         },
         "_id":"exampleaa",
         "_version":"exampleaa"
      },
      "_id":"exampleaa",
      "ts":1619695088
   }
]
}
```


Listing 4.6: Ukázka těla požadavku pro událost zařazení do spamu

Jako další případné využití služby Mandrill se nabízí použití statistik e-mailového provozu jako například e-mailové reputace dané organizace. Součástí monitorování by mohlo být i řešení případných výjimek či řešení nedoručitelných (chybných) e-mailů, které se během odesílání mohly objevit.

4.3 Budoucnost prototypu a e-mailového systému

Jak už bylo zmíněno v kapitole o prototypu, řešení je navrženo tak, aby mohlo být používáno podle potřeby v rámci celého portálu. Z toho důvodu bude prototyp na vytváření úloh představen i ostatním programátorům pracujícím nad portálem. Další využití prototypu by záleželo na požadavcích kladených na portál pohodlne.info, či jeho případné rozšíření. Po dohodě s konzultantem a vedením vývoje jsme rozhodli, že dalším krokem systému pro e-mailovou komunikaci bude převod do ostré

verze portálu. Než ale dojde ke zveřejnění systému, je nutné provést rozsáhlejší testování systému, které nebylo možné v rámci praxe realizovat. Testoval jsem systém pouze na jedné ze stránek portálu, kde jsem odhalil pouze drobné chyby, které jsem následně opravil. E-mailový systém je navržen tak, aby ho bylo možné použít odkudkoliv i bez použití úloh na pozadí. Jedním z požadavků v zadání bylo řízení dávkování komunikace jednotlivých e-mailů, to se ovšem po analýze a konzultaci ukázalo nad původní očekávání, a proto tento požadavek bude implementovaný až následně.

Jelikož ve firmě budu pracovat i nadále, hodlám pokračovat ve vylepšení a udržování systému. Další fází je potencionální nahrazení původní třídy `EmailCommunicationBean`, či jejího začlenění do třídy `EmailCommunicationSender`. Druhou možností rozšíření systému i prototypu je použití technologie `ServiceBuilder` nabízené společností Liferay. Tato technologie nabízí mnoho výhod při práci se třídami v rámci samotného portálu.

Dalším rozšířením se nabízí vytvoření pomocného systému pro zobrazování průběhu e-mailové komunikace a její další správy vycházející z již vytvořené stránky. Uživatel by mohl být například upozorněn pomocí vyskakovacího dialogu nebo by mohl dostat notifikaci na mobilní zařízení.

Kapitola 5

Využité dovednosti získané v průběhu studia

V rámci celé odborné praxe bylo nutné uplatnit již získané znalosti a dovednosti týkající se vytváření návrhu softwarového produktu a jeho následné implementace. Využil jsem své znalosti programovacího jazyka Java (z předmětů *Programovací jazyky I* a *Tvorba aplikací pro mobilní zařízení II*), které jsem v rámci praxe více prohloubil. Během praxe jsem mohl aplikovat i znalosti značkovacích jazyků pro psaní webových stránek (z předmětu *Vývoj internetových aplikací*). Samozřejmě jsem využíval i teoretické znalosti o správné struktuře a psaní kódu, kterými se zabývají teoretičtější předměty jako *Algoritmy I*, *II* a *Programování I*.

Při vytváření návrhu prototypu a e-mailového systému jsem využil své zkušenosti z oblasti práce s UML diagramy a navrhování informačních systémů (*Úvod do softwarového inženýrství* a *Vývoj informačních systémů*). Nesmírně důležitou součástí návrhu bylo i použití přístupu objektově orientovaného programování (*Programování II* a *Základy počítačové grafiky*), na kterém jsou postavené oba návrhy řešení.

Bohužel se z důvodu nenaplnění kapacity v rámci mého ročníku neotevřel předmět *Java Technologie*, jehož náplní je právě problematika technologií JavaEE, přesněji využití Enterprise JavaBeans a technologie JSF. Chybějící znalosti jsem byl nucen nastudovat sám, ale využil jsem podrobné dokumentace ze stránek předmětu.

Obecně mohu říct, že na vykonávání praxe jsem byl připraven a mé znalosti byly dostačující. Měl jsem pevné základy pro návrh a vývoj softwaru, ale chyběly mi nutné praktické zkušenosti programování složitějších systémů. Během bakalářského studia mi chyběl větší projekt, který by vyžadoval nutnou práci v kolektivu a vývoj podle podrobnějších uživatelských požadavků. Dále bych uvítal větší znalosti verzovacích systémů, které jsou v praxi standardem. Těmi se ale bohužel zabývá až semestrální projekt v rámci navazujícího magisterského studia.

Kapitola 6

Závěr

Praxe pro mě byla velkou zkušeností, u které jsem mohl otestovat své teoretické i praktické znalosti. S volbou bakalářské práce ve formě odborné praxe jsem spokojen. V obou částech jsem měl prostor pro implementaci vlastního řešení, u kterého jsem mohl přijít s novými požadavky na systém. Ze začátku jsem s novým vývojovým prostředím měl problémy, ale kolegové ve firmě mi byli vždy k dispozici. Ve zmíněné firmě pracuji už 5 let, ale poprvé jsem se stal opravdovou součástí vývojového týmu. Bohužel z důvodu pandemické situace byla firma nucena zavést home office, tudíž veškerou komunikaci s kolegy jsem musel řešit vzdáleně. To s sebou neslo občasné komplikace ve špatném vzájemném porozumění.

Mohl jsem si vyzkoušet návrh části reálného systému a jeho následnou implementaci, u které jsem ihned poznal případné nedostatky svého řešení. Znovu jsem si potvrdil, jak důležité je mít kvalitní a podrobný návrh, než začne samotné programování.

V praxi jsem poznal jak silný, ale při pochopení snadný nástroj může technologie Liferay být. Liferay nabízí nespočet kvalitních volně přístupných knihoven a frameworků, které je možné při potřebě využít. Tím, že je Liferay běžně používanou technologií, je možné využít rozsáhlých komunitních fór, kde je možné najít řešení i na konkrétnější problémy. Hodlám ve vývoji i nadále pokračovat a rozšiřovat tak své zkušenosti v oblasti vývoje webových aplikací a návrhu informačních systémů.

Literatura

1. *VRK plus s.r.o.: Produkty* [online]. 2021 [cit. 2021-01-10]. Dostupné z: `https://www.vrk.cz/`.
2. *Pohodlne.info: O nás* [online]. 2021 [cit. 2021-01-10]. Dostupné z: `https://pohodlne.info/o-nas`.
3. *Liferay Portal* [online]. 2021 [cit. 2021-02-01]. Dostupné z: `https://portal.liferay.dev/`.
4. BERGSTEN, Hans. Indroducing JavaServer Faces. In: *JavaServer Faces*. Sebastopol: O'Reilly Media, Inc., 2004, vol. 1, s. 1–10. ISBN 0-596-00539-3.
5. *JavaServer Faces Technology: Oracle JavaServer Faces Technology* [online]. 2021 [cit. 2021-02-15]. Dostupné z: `https://www.oracle.com/java/technologies/javaserverfaces.html`.
6. KOCSIS, Dániel. *Liferay Community: Background Task - a.k.a the secret weapon behind the new asynchronous staging* [online]. 2015 [cit. 2021-04-01]. Dostupné z: `https://liferay.dev/blogs/-/blogs/background-task-a-k-a-the-secret-weapon-behind-the-new-asynchronous-staging`.

Příloha A

E-mailový Executor

```
package cz.vrk.pi.crm.views.backgroundtask;

import com.liferay.portal.kernel.backgroundtask.BackgroundTaskResult;
import com.liferay.portal.kernel.repository.model.FileEntry;
import com.liferay.portal.kernel.util.StringPool;
import com.liferay.portal.model.*;
import com.liferay.portal.service.*;
import cz.vrk.pi.crm.LovConstants;
import cz.vrk.pi.crm.functions.CRMFunctions;
import cz.vrk.pi.crm.model.model.*;
import cz.vrk.pi.crm.model.service.MemberLocalServiceUtil;
import cz.vrk.pi.crm.views.email.ContactHelper;
import cz.vrk.pi.crm.views.email.EmailCommunicationBean;
import java.io.Serializable;
import java.util.*;

public class EmailBackgroundTaskExecutor extends CustomBackgroundTaskExecutor {
    public static final String SUBJECT = "subject";
    public static final String CONTENT = "content";
    public static final String TEMPLATE = "templateUuId";
    public static final String ORGANIZATION = "organization";
    public static final String ORGANIZATION_ID = "organizationId";
    public static final String SENDER_NAME = "senderName";
    public static final String SENDER_EMAIL = "senderEmail";
    public static final String USER_ID = "userId";
    public static final String RECIPIENT_LIST = "recipientList";
```

```
public static final String MEMBER_LIST = "memberList";
public static final String RECIPIENT_LIST_CC = "recipientListCc";
public static final String RECIPIENT_LIST_BCC = "recipientListBcc";
public static final String RECIPIENT_LIST_OTHER = "recipientListOther";
public static final String REPLY_TO = "replyTo";
public static final String TYPE_LIST = "typeList";
private EmailCommunicationSender emailCommunicationSender;

public static CustomBackgroundTaskService.ExecutorDefinition definition(
    String senderEmail, String senderName, String replayTo, String templateUId,
     ArrayList<Long> memberIds,ArrayList<String> selectRecipients, ArrayList<
    String> ccs, ArrayList<String> bccs, Long userId, Long organizationId) {
    return new CustomBackgroundTaskService.ExecutorDefinition() {
        @Override
        public Map<String, Serializable> getTaskContextMap() {
            Map<String, Serializable> map = new HashMap<String, Serializable>()
                ;
            map.put(SENDER_EMAIL, senderEmail);
            map.put(SENDER_NAME, senderName);
            map.put(REPLY_TO, replayTo);
            map.put(TEMPLATE, templateUId);
            map.put(MEMBER_LIST, memberIds);
            map.put(TYPE_LIST, selectRecipients);
            map.put(RECIPIENT_LIST_CC, ccs);
            map.put(RECIPIENT_LIST_BCC, bccs);
            map.put(USER_ID, userId);
            map.put(ORGANIZATION_ID, organizationId);
            return map;
            }

        @Override
        public Class<? extends CustomBackgroundTaskExecutor> getExecutorClass()
             {
            return EmailBackgroundTaskExecutor.class;
        }
    };
}
```

```
public EmailBackgroundTaskExecutor(){
    super();
}

@Override
public BackgroundTaskResult execute(BackgroundTask backgroundTask) throws
    Exception {
    //Inicialization of Execution
    sendInitialMessage(backgroundTask);

    this.emailCommunicationSender = new EmailCommunicationSender();

    //Getting all necessary data
    String senderName = (String) backgroundTask.getTaskContextMap().get(
        SENDER_NAME);
    String senderEmail = (String) backgroundTask.getTaskContextMap().get(
        SENDER_EMAIL);
    String replyTo = (String) backgroundTask.getTaskContextMap().get(REPLY_TO)
        ;

    ArrayList<Integer> memberIds = (ArrayList<Integer>) backgroundTask.
        getTaskContextMap().get(MEMBER_LIST);
    List<String> recipientTypes = (ArrayList<String>) backgroundTask.
        getTaskContextMap().get(TYPE_LIST);

    ArrayList<String> recipientCcMap = (ArrayList<String>) backgroundTask.
        getTaskContextMap().get(RECIPIENT_LIST_CC);
    ArrayList<String> recipientBccMap = (ArrayList<String>) backgroundTask.
        getTaskContextMap().get(RECIPIENT_LIST_BCC);
    ArrayList<String> recipientOthersMap = (ArrayList<String>) backgroundTask.
        getTaskContextMap().get(RECIPIENT_LIST_OTHER);

    String templateUuId = (String) backgroundTask.getTaskContextMap().get(
        TEMPLATE);
    String subject = (String) backgroundTask.getTaskContextMap().get(SUBJECT);
    String content = (String) backgroundTask.getTaskContextMap().get(CONTENT);
```

```
User user = UserLocalServiceUtil.fetchUser(Long.valueOf((Integer)
    backgroundTask.getTaskContextMap().get(USER_ID)));
Organization organization = OrganizationLocalServiceUtil.fetchOrganization
    (Long.valueOf((Integer)backgroundTask.getTaskContextMap().get(
    ORGANIZATION_ID)));
CRMFunctions crmFunctions = this.emailCommunicationSender.getCrmFunctions(
    organization, user.getLocale());

List<FileEntry> backgroundTaskAttachments = backgroundTask.
    getAttachmentsFileEntries();

//Getting all necessary components from data

List<Member> members = new ArrayList<>();
for(Integer id : memberIds){
    members.add(MemberLocalServiceUtil.fetchMember(Long.valueOf(id)));
}

List<MemberCommunicationEmail> ccList = this.emailCommunicationSender.
    getMemberCommunicationEmails(recipientCcMap, LovConstants.MCET_CC);
List<MemberCommunicationEmail> bccList = this.emailCommunicationSender.
    getMemberCommunicationEmails(recipientBccMap, LovConstants.MCET_BCC);

if(recipientOthersMap!=null){
    bccList = this.emailCommunicationSender.getMemberCommunicationEmails(
        recipientOthersMap, LovConstants.MCET_BCC);
}
EmailCommunicationBean emailCommunicationBean = this.
    emailCommunicationSender.getEmailCommunicationBean(crmFunctions);

//Sending itself
int emailCounter = 0;

for(Member member : members){
    MemberCommunication memberCommunication = this.emailCommunicationSender
        .initMemberCommunication();
    if(senderEmail!=null && !senderEmail.isEmpty()) {
        if(senderName!=null) {
```

```
            memberCommunication.setEmailFrom(senderEmail, senderName);
            memberCommunication.setFrom(senderName);
        }
        else{
            memberCommunication.setEmailFrom(senderEmail, "");
        }
    }
    else {
        throw new Exception("Email adress is empty");
    }
    if(!replyTo.equals(StringPool.BLANK)){
        memberCommunication.setReplyTo(replyTo);
    }
    memberCommunication.setMember(member.getId());

    if(templateUuId != null && !templateUuId.isEmpty()) {
        this.emailCommunicationSender.setContent(templateUuId, organization
            .getOrganizationId(), memberCommunication, crmFunctions);
    }
    else{
        memberCommunication.setSubject(subject);
        memberCommunication.setContent(content);
    }
    //Getting and converting all Member's contacts
    List<ContactHelper> contacts = this.emailCommunicationSender.
        getAllContacts(member);
    List<ContactHelper> filteredContacts = contacts;
    if(recipientTypes!=null) {
        filteredContacts = this.emailCommunicationSender.
            filterContactByString(contacts, recipientTypes);
    }
    List<MemberCommunicationEmail> recipientList = this.
        emailCommunicationSender.getMemberCommunicationEmails(
        filteredContacts);

    this.emailCommunicationSender.setRecipients(memberCommunication,
        recipientList);
    this.emailCommunicationSender.setCc(memberCommunication, ccList);
```

```
            this.emailCommunicationSender.setBcc(memberCommunication, bccList);

            this.emailCommunicationSender.sendEmail(memberCommunication,
                backgroundTaskAttachments, emailCommunicationBean);

            //Sending message about progress
            emailCounter+= 1;
            float percentage = emailCounter*(100.0/members.size());
            sendMessage(backgroundTask, "In progress", percentage+"% done");
        }
        return setFinalResult(backgroundTask, true, "");
    }
}
```

Listing A.1: Kompletní kód emailového executoru